第1章 概述

1.1 简介

LPC2100/lLPC2105/LPC2106 系列微控制器是飞利浦半导体退出的基于 16/32 位 ARM7TDMI-S CPU，并带有 128/256 k 字节(kB)嵌入的高速 Flash 存储器的为控制器。128 位 宽度的存储器接口和独特的加速结构使 32 位代码能够在最大时钟速率下运行。对代码规模有严格控制的应用可使用 16 位 Thumb 模式将代码规模降低超过 30%，而性能的损失却很小。由于 LPC2100/lLPC2105/LPC2106 系列微控制器采用非常小的 64 脚封装、极低的功耗、多个 32 位定时器、4 路 10 位 ADC、PWM 输出以及多达 9 个外部中断，这使它们特别适用于工业控制、医疗系统、访问控制和电子收款机(POS)等应用领域。由于内置了宽范围的串行通信接口，它们也非常适合于通信网关、协议转换器、嵌入式软件调制解调器以及其它各种类型的应用。

因为 LPC2100/lLPC2105/LPC2106 系列微控制器没有外部总线控制器，所以它们外接扩展芯片不是很方便。不过，因为它们的速度很快，所以即使使用软件模拟总线外接扩展芯片也比普通的 80c51 快得多，而丰富的片内资源也不是普通 51 能够比拟的。可以预见，如果其价格大幅度降低，它们将对高端 8 位单片机产生极大压力。

为了方便用户使用 LPC2100/lLPC2105/LPC2106 外接扩展芯片，本公司特别开发了 ZLG/VBUS for LPC210x 软件包，本文档就是这个软件包的使用说明书。

1.2 特点

- 兼容 51 时序，方便用户从 51 过渡到 ARM
- ALE、RD、WR 可以定义到任意的 GPIO，总线只需要连续 GPIO
- 8 位数据总线，地址可以配置为 8 位、16 位或 24 位
- 8 位数据总线仅占用 11 个 GPIO、16 位和 24 位仅占用 19 个 GPIO
- 支持多任务操作系统
- 公开源代码

注：24 位地址还没有进行严格测试。

1.3 注意事项

复用的数据/地址低 8 位平时为输出状态，与标准的 80C51 时序的高阻态不一样。这样做是为了加快总线时速度。

第2章 使用

2.1 8 位方式的电路连接示意图及例子

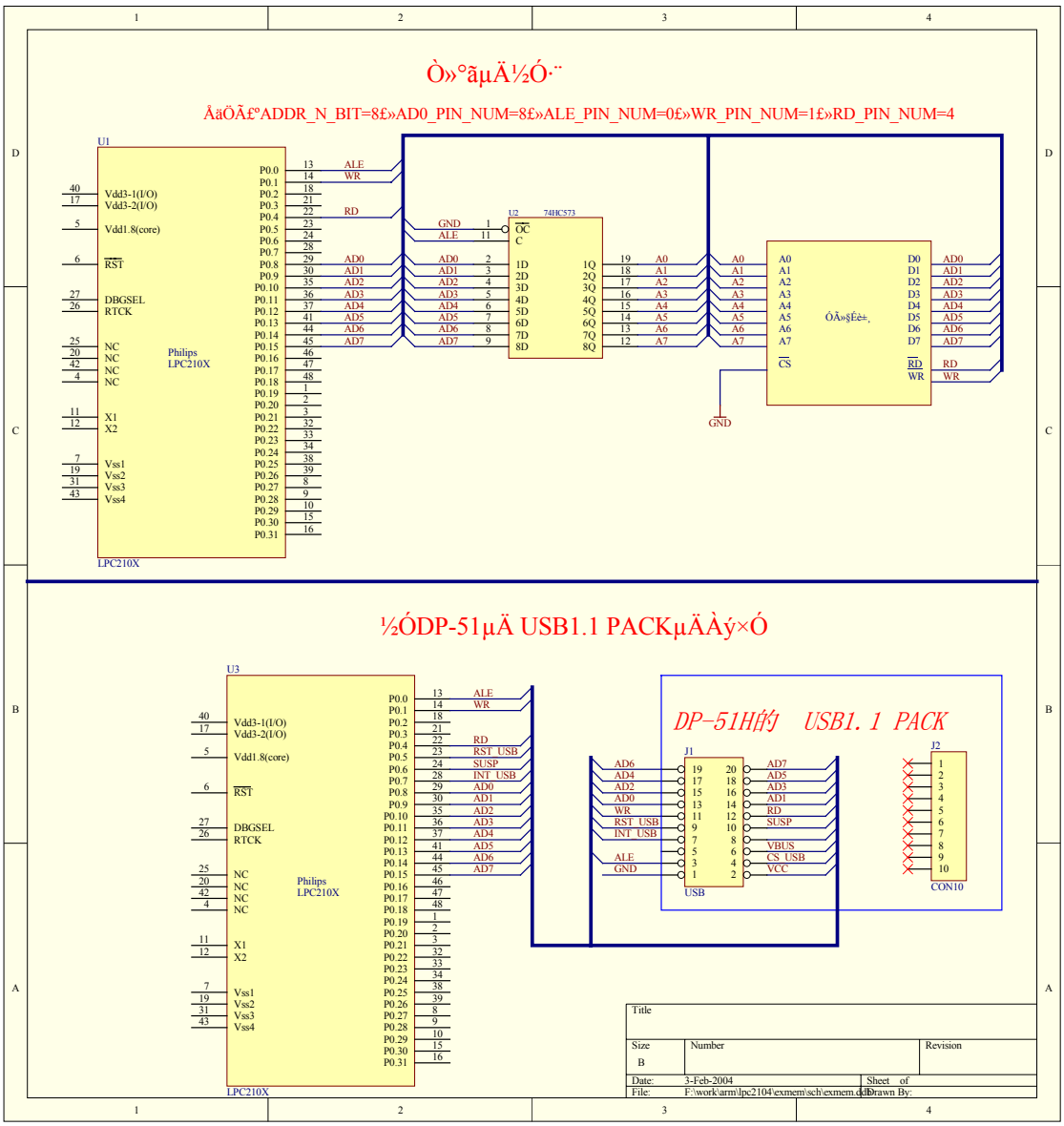

图 2.1 8 位地址方式的电路连接示意图及例子

2.2 16 位方式的电路连接的例子

本软件包所附带的范例即使用此电路。

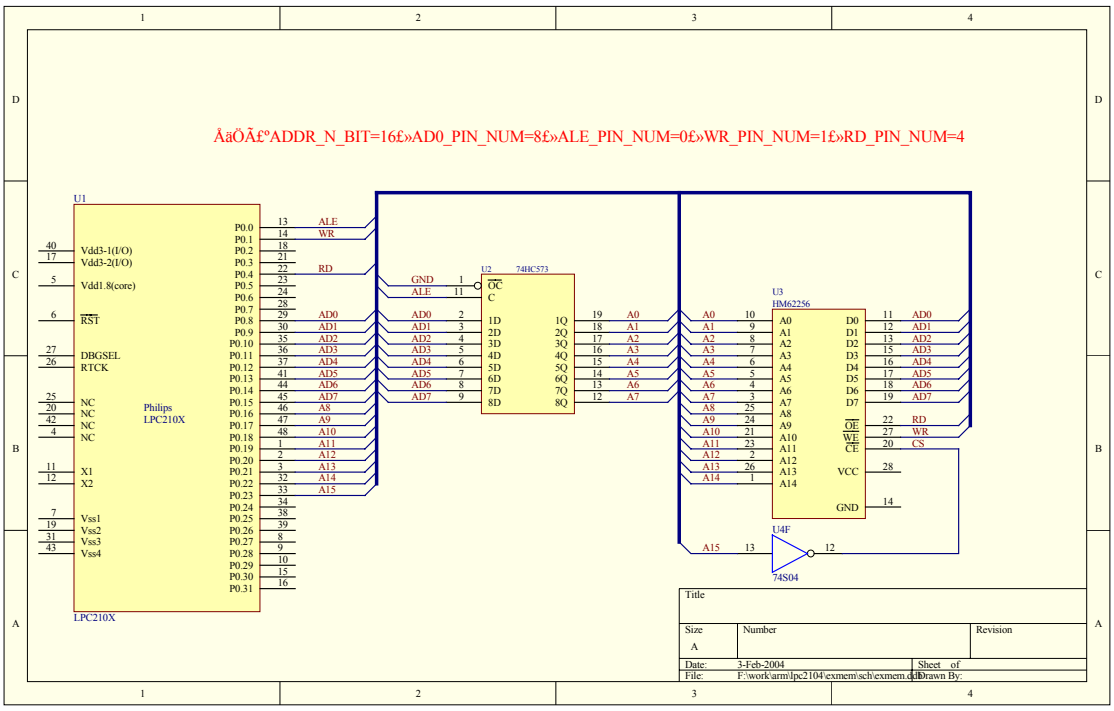

图 2.2 16 位地址方式的电路连接的例子

2.3 24 位方式的电路连接的示意图

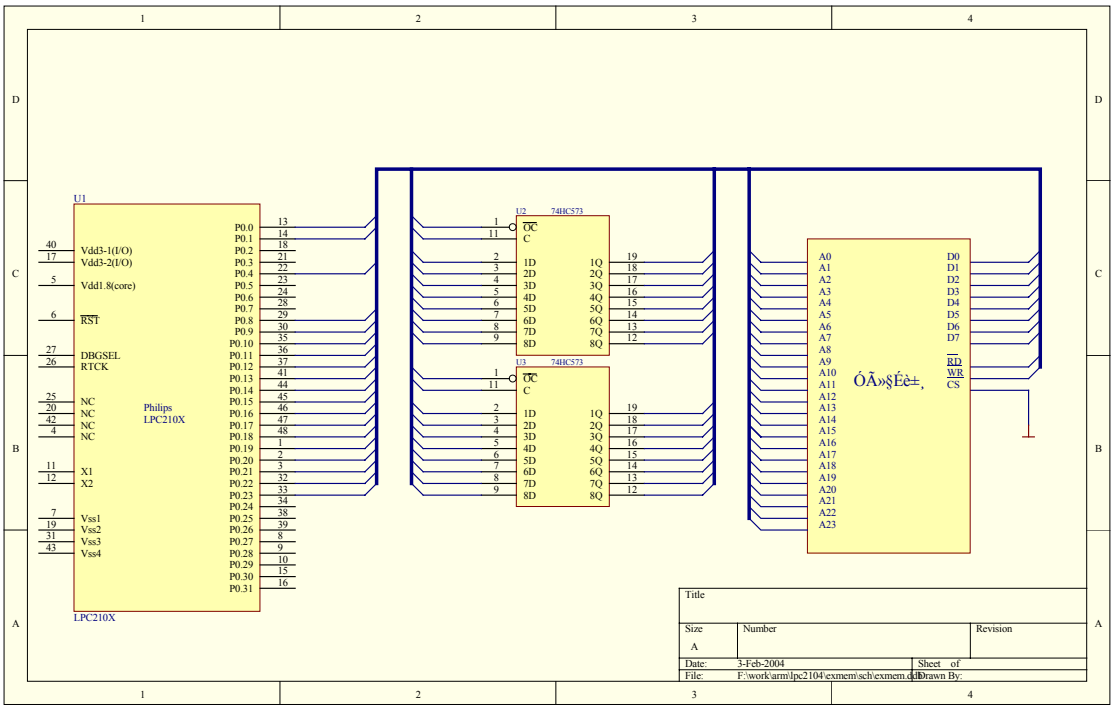

图 2.3 24 位地址方式的电路连接示意图

2.4 配置软件包使用的 GPIO

在使本软件包用户必须分配好总线使用的 GPIO。本软件包对 GPIO 的唯一限制是分配的地址总线（包括复用的数据总线）必须是连续的，而且有特定的顺序。其顺序参考图 2.1、图 2.2 和图 2.3。

配置软件包使用的 GPIO 必须在文件 config.h 中进行，其例子见程序清单 2.1。其中程序清单 2.1(1)设置地址总线的宽度，只能为 8、16 和 24 三个值之一。程序清单 2.1(2)设置信号 AD0 对应 GPIO，其中 0 对应 P0.0，1 对应 P0.1，以次类推。地址/数据总线的其它信号以这个信号为基准，依照图 2.1、图 2.2 和图 2.3 的顺序定义。当 ADDR_N_BIT=8 时，AD0_PIN_NUM 不能大于 24；否则 AD0_PIN_NUM 不能大于 16。程序清单 2.1(3) 信号 ALE 对应 GPIO，其中 0 对应 P0.0，1 对应 P0.1，以次类推。程序清单 2.1(4) 信号 WR 对应 GPIO，其中 0 对应 P0.0，1 对应 P0.1，以次类推。程序清单 2.1(5) 信号 RD 对应 GPIO，其中 0 对应 P0.0，1 对应 P0.1，以次类推。*注意：程序清单 2.1(5)必须放在程序清单 2.1(1)~(5)的后面，否则编译错误。*

程序清单 2.1 配置总线使用的 GPIO

```
#define ADDR_N_BIT          16                                  (1)
#define AD0_PIN_NUM         8                                   (2)
#define ALE_PIN_NUM         0                                   (3)
#define WR_PIN_NUM          1                                   (4)
#define RD_PIN_NUM          4                                   (5)

#include "exbus.h"                                              (6)
```

2.5 OS_ENTER_CRITICAL()和 OS_EXIT_CRITICAL()

因为一个总线读时序和写操作是不可分割的整体，所以本软件模块的读写函数是不可重入的。如果用户在中断和中程序均需要访问模拟总线器件（或多个中断要访问模拟总线器件），或是在多任务环境下不只一个任务何/或中断要访问模拟总线器件，则必须在在访问模拟总线器件前关中断，在访问模拟总线器件后开中断。为了避免不同环境下开关中断的方法不一样，本函数使用 OS_ENTER_CRITICAL 关中断，用 OS_EXIT_CRITICAL 开中断。其例子见程序清单 2.2。这段代码在 config.h 中。其中宏 IN_EXBUS 是本软件模块定义的宏，使用它使这段代码仅在本软件模块内有效。

程序清单 2.2 开、关中断的例子

```
#ifdef IN_EXBUS
#define     NoInt           0x80
/* 关中断 */
    __inline OS_ENTER_CRITICAL(void)
{
    __asm
    {
        MRS     R0, CPSR
        ORR     R0, R0, #NoInt
        MSR     CPSR_c, R0
    }
}
/* 开中断 */
    __inline OS_EXIT_CRITICAL(void)
{
    __asm
```

```
    {
        MRS     R0, CPSR
        BIC     R0, R0, #NoInt
        MSR     CPSR_c, R0
    }
}
#endif
```

2.6 初始化模拟总线

在使用模拟总线前必须初始化它，这是通过调用函数 ExBusInit()实现。函数 ExBusInit()没有参数和返回值。

表 2.1 ExBusInit 函数

函数名称	ExBusInit	所属文件	exbus.c
函数原型	void ExBusInit(ADDR ExAddr)		
功能描述	初始化它模拟总线		
函数参数	无		
函数返回值	无		
特殊说明和注意点	无		

2.7 从外部器件中读取数据

初始化模拟总线后可以调用函数 ExBusRead()读取数据。函数 ExBusRead()有唯一参数指示需要读取的地址，返回值为读到的数据。

表 2.2 ExBusRead 函数

函数名称	ExBusRead	所属文件	exbus.c
函数原型	uint8 ExBusRead(ADDR ExAddr)		
功能描述	从总线中读取数据		
函数参数	无		
函数返回值	无		
特殊说明和注意点	无		

2.8 写数据到外部器件

初始化模拟总线后可以调用函数 ExBusWrite()写数据到数据总线。函数 ExBusWrite ()有两个参数指示

需要读取的地址和需要写的数据，它没有返回值。

表 2.3 ExBusWrite 函数

函数名称	ExBusWrite	所属文件	exbus.c
函数原型	uint8 ExBusRead(ADDR ExAddr)		
功能描述	写数据到总线		
函数参数	ExAddr: 将要写入的地址，Data: 将要写入的数据		
函数返回值	无		
特殊说明和注意点	无		

2.9 使用范例

这个例子的电路原理图参考图 2.2。程序用来检测 RAM，如果 RAM 的某一字节有错误，程序会停留在中间某处。此时，循环变量 I 指示出出错的地址。程序比较简单，读者自己分析一下。

程序清单 2.3 使用范例

```
#include "config.h"

const uint8 WriteData[] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80,
                           0x55, 0xaa, 0xff, 0x00};
uint8 temp;
uint8  j;
uint32 i;

        int main (void)
{
    ExBusInit();
    for (i = 0x8000; i < 0x10000; i++)
    {
        ExBusWrite(i,0);
    }
    for (i = 0x8000; i < 0x10000; i++)
    {
        temp = ExBusRead(i);
        if (temp != 0)
        {
            while(1)
            {
            }
```

```
            }
            j = 0;
            while (1)
            {
                ExBusWrite(i,WriteData[j]);
                temp = ExBusRead(i);
                if (temp != WriteData[j])
                {
                    while (1);
                    {
                    }
                }
                j++;
                if (WriteData[j] == 0)
                {
                    break;
                }
            }
        }
        while (1);

        return 0;
}
```